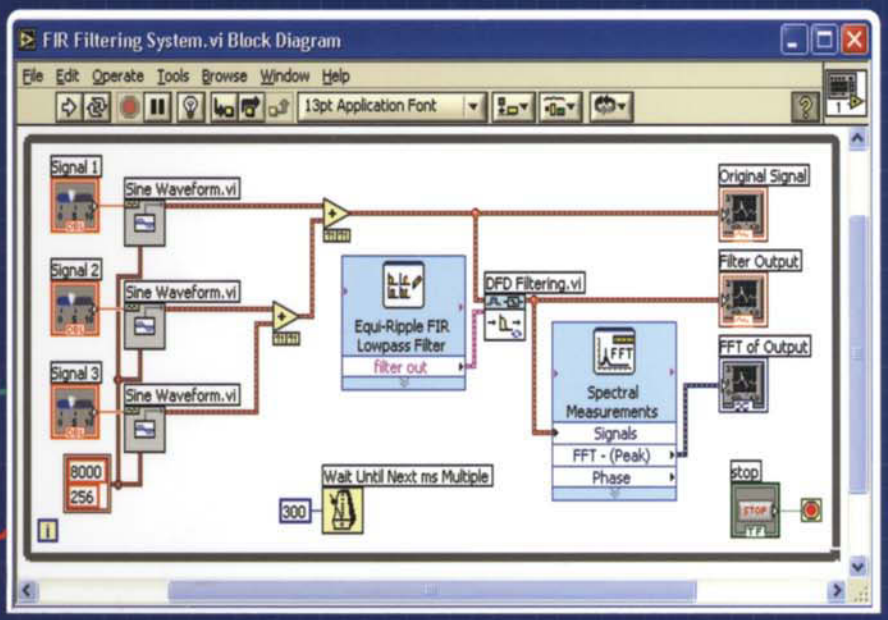
ELSEVIER

Includes CD with lab exercises and project examples

Digital Signal Processing System-Level Design

Using LabVIEW™



NASSER KEHTARNAVAZ and NAMJIN KIM

Newnes

Digital Signal Processing System-Level Design Using LabVIEW™

This Page Intentionally Left Blank

Digital Signal Processing System-Level Design Using LabVIEW™

by Nasser Kehtarnavaz and Namjin Kim
University of Texas at Dallas

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO
Newnes is an imprint of Elsevier

Newnes is an imprint of Elsevier
30 Corporate Drive, Suite 400, Burlington, MA 01803, USA
Linacre House, Jordan Hill, Oxford OX2 8DP, UK

Copyright © 2005, Elsevier Inc. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.

Permissions may be sought directly from Elsevier's Science & Technology Rights Department in Oxford, UK: phone: (+44) 1865 843830, fax: (+44) 1865 853333, e-mail: permissions@elsevier.com.uk. You may also complete your request on-line via the Elsevier homepage (http://elsevier.com), by selecting "Customer Support" and then "Obtaining Permissions."

Recognizing the importance of preserving what has been written, Elsevier prints its books on acid-free paper whenever possible.

Library of Congress Cataloging-in-Publication Data

(Application submitted)

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library.

ISBN: 0-7506-7914-X

For information on all Newnes publications
visit our Web site at www.books.elsevier.com

05 06 07 08 09 10 10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

Working together to grow
libraries in developing countries
www.elsevier.com | www.bookaid.org | www.sabre.org
ELSEVIER BOOK AID International Sabre Foundation

Contents

Preface ..... ***ix***

What's on the CD-ROM? ..... ***xi***

Chapter 1: Introduction ***1***

1.1 Digital Signal Processing Hands-On Lab Courses 2
1.2 Organization 3
1.3 Software Installation 3
1.4 Updates 4
1.5 Bibliography 4

Chapter 2: LabVIEW Programming Environment ***5***

2.1 Virtual Instruments (VIs) 5
2.2 Graphical Environment 7
2.3 Building a Front Panel 8
2.4 Building a Block Diagram 10
2.5 Grouping Data: Array and Cluster 12
2.6 Debugging and Profiling VIs 13
2.7 Bibliography 14

Lab 1: Getting Familiar with LabVIEW: Part I ***15***

L1.1 Building a Simple VI 15
L1.2 Using Structures and SubVIs 23
L1.3 Create an Array with Indexing 27
L1.4 Debugging VIs: Probe Tool 28
L1.5 Bibliography 30

Lab 2: Getting Familiar with LabVIEW: Part II ***31***

L2.1 Building a System VI with Express VIs 31
L2.2 Building a System with Regular VIs 37
L2.3 Profile VI 41
L2.4 Bibliography 42

Chapter 3: Analog-to-Digital Signal Conversion 43
3.1 Sampling 43
3.2 Quantization 49
3.3 Signal Reconstruction 51
Lab 3: Sampling, Quantization and Reconstruction 55
L3.1 Aliasing 55
L3.2 Fast Fourier Transform 59
L3.3 Quantization 64
L3.4 Signal Reconstruction 68
L3.5 Bibliography 72
Chapter 4: Digital Filtering 73
4.1 Digital Filtering 73
4.2 LabVIEW Digital Filter Design Toolkit 77
4.3 Bibliography 78
Lab 4: FIR/IIR Filtering System Design 79
L4.1 FIR Filtering System 79
L4.2 IIR Filtering System 85
L4.3 Building a Filtering System Using Filter Coefficients 90
L4.4 Filter Design Without Using DFD Toolkit 91
L4.5 Bibliography 94
Chapter 5: Fixed-Point versus Floating-Point 95
5.1 Q-format Number Representation 95
5.2 Finite Word Length Effects 99
5.3 Floating-Point Number Representation 100
5.4 Overflow and Scaling 102
5.5 Data Types in LabVIEW 102
5.6 Bibliography 104
Lab 5: Data Type and Scaling 105
L5.1 Handling Data types in LabVIEW 105
L5.2 Overflow Handling 107
L5.3 Scaling Approach 111
L5.4 Digital Filtering in Fixed-Point Format 113
L5.5 Bibliography 122
Chapter 6: Adaptive Filtering 123
6.1 System Identification 123
6.2 Noise Cancellation 124
6.3 Bibliography 126
Lab 6: Adaptive Filtering Systems 127
L6.1 System Identification 127
L6.2 Noise Cancellation 134

L6.3 Bibliography 138
Chapter 7: Frequency Domain Processing 139
7.1 Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT) 139
7.2 Short-Time Fourier Transform (STFT) 140
7.3 Discrete Wavelet Transform (DWT) 142
7.4 Signal Processing Toolset 144
7.5 Bibliography 145
Lab 7: FFT, STFT and DWT 147
L7.1 FFT versus STFT 147
L7.2 DWT 152
L7.3 Bibliography 156
Chapter 8: DSP Implementation Platform: TMS320C6x Architecture and Software Tools 157
8.1 TMS320C6X DSP 157
8.2 C6x DSK Target Boards 161
8.3 DSP Programming 163
8.4 Bibliography 166
Lab 8: Getting Familiar with Code Composer Studio 167
L8.1 Code Composer Studio 167
L8.2 Creating Projects 167
L8.3 Debugging Tools 173
L8.4 Bibliography 182
Chapter 9: LabVIEW DSP Integration 183
9.1 Communication with LabVIEW: Real-Time Data Exchange (RTDX) 183
9.2 LabVIEW DSP Test Integration Toolkit for TI DSP 183
9.3 Combined Implementation: Gain Example 184
9.4 Bibliography 190
Lab 9: DSP Integration Examples 191
L9.1 CCS Automation 191
L9.2 Digital Filtering 193
L9.3 Fixed-Point Implementation 202
L9.4 Adaptive Filtering Systems 206
L9.5 Frequency Processing: FFT 211
L9.6 Bibliography 220
Chapter 10: DSP System Design: Dual-Tone Multi-Frequency (DTMF) Signaling 221
10.1 Bibliography 224
Lab 10: Dual-Tone Multi-Frequency 225
L10.1 DTMF Tone Generator System 225

L10.2 DTMF Decoder System 228
L10.3 Bibliography 230
Chapter 11: DSP System Design: Software-Defined Radio 231
11.1 QAM Transmitter 231
11.2 QAM Receiver 234
11.3 Bibliography 238
Lab 11: Building a 4-QAM Modem 239
L11.1 QAM Transmitter 239
L11.2 QAM Receiver 242
L11.3 Bibliography 252
Chapter 12: DSP System Design: MP3 Player 253
12.1 Synchronization Block 254
12.2 Scale Factor Decoding Block 256
12.3 Huffman Decoder 257
12.4 Requantizer 259
12.5 Reordering 261
12.6 Alias Reduction 261
12.7 IMDCT and Windowing 262
12.8 Polyphase Filter Bank 264
12.9 Bibliography 266
Lab 12: Implementation of MP3 Player in LabVIEW 267
L12.1 System-Level VI 267
L12.2 LabVIEW Implementation 268
L12.3 Modifications to Achieve Real-Time Decoding 281
L12.4 Bibliography 286
Index 287

Preface

For many years, I have been teaching DSP (Digital Signal Processing) lab courses using various TI (Texas Instruments) DSP platforms. One question I have been getting from students in a consistent way is, "Do we have to know C to take DSP lab courses?" Until last year, my response was, "Yes, C is a prerequisite for taking DSP lab courses." However, last year for the first time, I provided a different response by saying, "Though preferred, it is not required to know C to take DSP lab courses." This change in my response came about because I started using LabVIEW to teach students how to design and analyze DSP systems in our DSP courses.

The widely available graphical programming environments such as LabVIEW have now reached the level of maturity that allow students and engineers to design and analyze DSP systems with ease and in a relatively shorter time as compared to C and MATLAB. I have observed that many students taking DSP lab courses, in particular at the undergraduate level, often struggle and spend a fair amount of their time debugging C and MATLAB code instead of placing their efforts into understanding signal processing system design issues. The motivation behind writing this book has thus been to avoid this problem by adopting a graphical programming approach instead of the traditional and commonly used text-based programming approach in DSP lab courses. As a result, this book allows students to put most of their efforts into building DSP systems rather than debugging C code when taking DSP lab courses.

One important point that needs to be mentioned here is that in order to optimize signal processing algorithms on a DSP processor, it is still required to know and use C and/or assembly programming. The existing graphical programming environments are not meant to serve as optimizers when implementing signal processing algorithms on DSP processors or other hardware platforms. This point has been addressed

in this book by providing two chapters which are dedicated solely to algorithm implementation on the TI family of TMS320C6000 DSP processors.

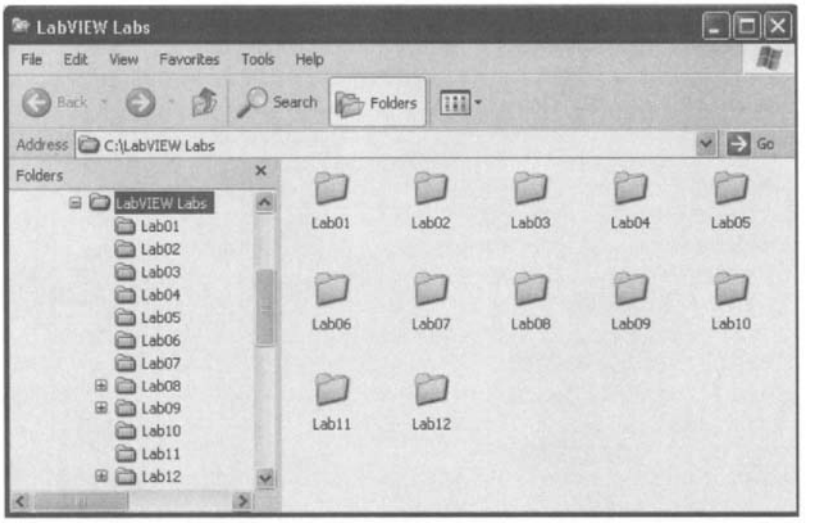
It is envisioned that this alternative graphical programming approach to designing digital signal processing systems will allow more students to get exposed to the field of DSP. In addition, the book is written in such a way that it can be used as a self-study guide by DSP engineers who wish to become familiar with LabVIEW and use it to design and analyze DSP systems.

I would like to express my gratitude to NI (National Instruments) for their support of this book. In particular, I wish to thank Jim Cahow, Academic Resources Manager at NI, and Ravi Marawar, Academic Program Manager at NI, for their valuable feedback. I am pleased to acknowledge Chuck Glaser, Senior Acquisition Editor at Elsevier, and Cathy Wicks, University Program Manager at TI, for their promotion of the book. Finally, I am grateful to my family who put up with my preoccupation on this book-writing project.

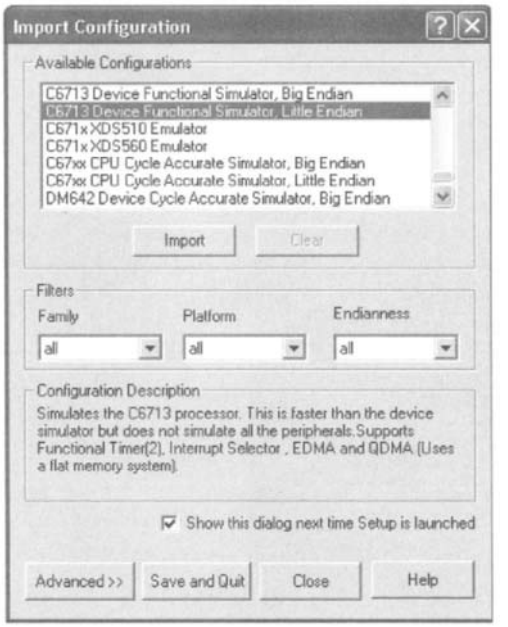
Nasser Kehtarnavaz
December 2004

What's on the CD-ROM?

- The accompanying CD-ROM includes all the lab files discussed throughout the book. These files are placed in corresponding folders as follows:
  - Lab01: Getting familiar with LabVIEW: Part I
  - Lab02: Getting familiar with LabVIEW: Part II
  - Lab03: Sampling, Quantization, and Reconstruction
  - Lab04: FIR/IIR Filtering System Design
  - Lab05: Data Type and Scaling
  - Lab06: Adaptive Filtering Systems
  - Lab07: FFT, STFT, and DWT
  - Lab08: Getting Familiar with Code Composer Studio
  - Lab09: DSP Integration Examples
  - Lab10: Building Dual Tone Multi Frequency System in LabVIEW
  - Lab11: Building 4-QAM Modem System in LabVIEW
  - Lab12: Building MP3 Player System in LabVIEW
- To run the lab files, the National Instruments LabVIEW 7.1 is required and assumed installed. The lab files need to be copied into the folder "C:\LabVIEW Labs\".



- For Lab 8 and Lab 9, the Texas Instruments Code Composer Studio 2.2 (CCStudio) is required and assumed installed in the folder "C:\ti\". The subfolders correspond to the following DSP platforms:
  - DSK 6416
  - DSK 6713
  - Simulator (configured as DSK6713 as shown below)


Import Configuration
Available Configurations
C6713 Device Functional Simulator, Big Endian
C6713 Device Functional Simulator, Little Endian
C671x XDS510 Emulator
C671x XDS560 Emulator
C67xx CPU Cycle Accurate Simulator, Big Endian
C67xx CPU Cycle Accurate Simulator, Little Endian
DM642 Device Cycle Accurate Simulator, Big Endian
Import
Clear
Filters
Family
Platform
Endianness
all
all
all
Configuration Description
Simulates the C6713 processor. This is faster than the device simulator but does not simulate all the peripherals. Supports Functional Timer(2), Interrupt Selector , EDMA and QDMA (Uses a flat memory system).
Show this dialog next time Setup is launched
Advanced >>
Save and Quit
Close
Help

CHAPTER 1

Introduction

The field of digital signal processing (DSP) has experienced a considerable growth in the last two decades, primarily due to the availability and advancements in digital signal processors (also called DSPs). Nowadays, DSP systems such as cell phones and high-speed modems have become an integral part of our lives.

In general, sensors generate analog signals in response to various physical phenomena that occur in an analog manner (that is, in continuous time and amplitude). Processing of signals can be done either in the analog or digital domain. To perform the processing of an analog signal in the digital domain, it is required that a digital signal is formed by sampling and quantizing (digitizing) the analog signal. Hence, in contrast to an analog signal, a digital signal is discrete in both time and amplitude. The digitization process is achieved via an analog-to-digital (A/D) converter. The field of DSP involves the manipulation of digital signals in order to extract useful information from them.

There are many reasons why one might wish to process an analog signal in a digital fashion by converting it into a digital signal. The main reason is that digital processing allows programmability. The same processor hardware can be used for many different applications by simply changing the code residing in memory. Another reason is that digital circuits provide a more stable and tolerant output than analog circuits—for instance, when subjected to temperature changes. In addition, the advantage of operating in the digital domain may be intrinsic. For example, a linear phase filter or a steep-cutoff notch filter can easily be realized by using digital signal processing techniques, and many adaptive systems are achievable in a practical product only via digital manipulation of signals. In essence, digital representation (zeroes and ones) allows voice, audio, image, and video data to be treated the same for error-tolerant digital transmission and storage purposes.

1.1 Digital Signal Processing Hands-On Lab Courses

Nearly all electrical engineering curricula include DSP courses. DSP lab or design courses are also being offered at many universities concurrently or as follow-ups to DSP theory courses. These hands-on lab courses have played a major role in student understanding of DSP concepts. A number of textbooks, such as [1-3], have been written to provide the teaching materials for DSP lab courses. The programming language used in these textbooks consists of either C, MATLAB®, or Assembly, that is text-based programming. In addition to these programming skills, it is becoming important for students to gain experience in a block-based or graphical (G) programming language or environment for the purpose of designing DSP systems in a relatively short amount of time. Thus, the main objective of this book is to provide a block-based or system-level programming approach in DSP lab courses. The block-based programming environment chosen is LabVIEW™.

LabVIEW (Laboratory Virtual Instrumentation Engineering Workbench) is a graphical programming environment developed by National Instruments (NI), which allows high-level or system-level designs. It uses a graphical programming language to create so-called Virtual Instruments (VI) blocks in an intuitive flowchart-like manner. A design is achieved by integrating different components or subsystems within a graphical framework. LabVIEW provides data acquisition, analysis, and visualization features well suited for DSP system-level design. It is also an open environment accommodating C and MATLAB code as well as various applications such as ActiveX and DLLs (Dynamic Link Libraries).

This book is written primarily for those who are already familiar with signal processing concepts and are interested in designing signal processing systems without needing to be proficient C or MATLAB programmers. After familiarizing the reader with LabVIEW, the book covers a LabVIEW-based approach to generic experiments encountered in a typical DSP lab course. It brings together in one place the information scattered in several NI LabVIEW manuals to provide the necessary tools and know-how for designing signal processing systems within a one-semester structured course. This book can also be used as a self-study guide to design signal processing systems using LabVIEW.

In addition, for those interested in DSP hardware implementation, two chapters in the book are dedicated to executing selected portions of a LabVIEW designed system on an actual DSP processor. The DSP processor chosen is TMS320C6000. This processor is manufactured by Texas Instruments (TI) for computationally intensive signal processing applications. The DSP hardware utilized to interface with

LabVIEW is the TI's C6416 or C6713 DSK (DSP Starter Kit) board. It should be mentioned that since the DSP implementation aspect of the labs (which includes C programs) is independent of the LabVIEW implementation, those who are not interested in the DSP implementation may skip these two chapters. It is also worth pointing out that once the LabVIEW code generation utility becomes available, any portion of a LabVIEW designed system can be executed on this DSP processor without requiring any C programming.

1.2 Organization

The book includes twelve chapters and twelve labs. After this introduction, the LabVIEW programming environment is presented in Chapter 2. Lab 1 and Lab 2 in Chapter 2 provide a tutorial on getting familiar with the LabVIEW programming environment. The topic of analog to digital signal conversion is presented in Chapter 3, followed by Lab 3 covering signal sampling examples. Chapter 4 involves digital filtering. Lab 4 in Chapter 4 shows how to use LabVIEW to design FIR and IIR digital filters. In Chapter 5, fixed-point versus floating-point implementation issues are discussed followed by Lab 5 covering data type and fixed-point effect examples. In Chapter 6, the topic of adaptive filtering is discussed. Lab 6 in Chapter 6 covers two adaptive filtering systems consisting of system identification and noise cancellation. Chapter 7 presents frequency domain processing followed by Lab 7 covering the three widely used transforms in signal processing: fast Fourier transform (FFT), short time Fourier transform (STFT), and discrete wavelet transform (DWT). Chapter 8 discusses the implementation of a LabVIEW-designed system on the TMS320C6000 DSP processor. First, an overview of the TMS320C6000 architecture is provided. Then, in Lab 8, a tutorial is presented to show how to use the Code Composer Studio™ (CCStudio) software development tool to achieve the DSP implementation. As a continuation of Chapter 8, Chapter 9 and Lab 9 discuss the issues related to the interfacing of LabVIEW and the DSP processor. Chapters 10 through 12, and Labs 10 through 12, respectively, discuss the following three DSP systems or project examples that are fully designed via LabVIEW: (i) dual-tone multi-frequency (DTMF) signaling, (ii) software-defined radio, and (iii) MP3 player.

1.3 Software Installation

LabVIEW 7.1, which is the latest version at the time of this writing, is installed by running *setup.exe* on the LabVIEW 7.1 Installation CD. Some lab portions use the LabVIEW toolkits 'Digital Filter Design,' 'Advanced Signal Processing,' and 'DSP

Test Integration for TI DSP.' Each of these toolkits can be installed by running *setup.exe* located on the corresponding toolset CD.

If one desires to run parts of a LabVIEW designed system on a DSP processor, then it is necessary to install the Code Composer Studio software tool. This is done by running *setup.exe* on the CCStudio CD. The most updated version of CCStudio at the time of this writing, CCStudio 2.2, is used in the DSK-related labs.

The accompanying CD includes all the files necessary for running the labs covered throughout the book.

1.4 Updates

Considering that any programming environment goes through enhancements and updates, it is expected that there will be updates of LabVIEW and its toolkits. To accommodate for such updates and to make sure that the labs provided in the book can still be used in DSP lab courses, any new version of the labs will be posted at the website http://www.utdallas.edu/~kehtar/LabVIEW for easy access. It is recommended that this website be periodically checked to download any necessary updates.

1.5 Bibliography

[1] N. Kehtarnavaz, *Real-Time Digital Signal Processing Based on the TMS320C6000*, Elsevier, 2005.

[2] S. Kuo and W-S. Gan, *Digital Signal Processors: Architectures, Implementations, and Applications*, Prentice-Hall, 2005.

[3] R. Chassaing, *DSP Applications Using C and the TMS320C6x DSK*, Wiley Inter-Science, 2002.

CHAPTER 2

LabVIEW Programming Environment

LabVIEW constitutes a graphical programming environment that allows one to design and analyze a DSP system in a shorter time as compared to text-based programming environments. LabVIEW graphical programs are called virtual instruments (VIs). VIs run based on the concept of data flow programming. This means that execution of a block or a graphical component is dependent on the flow of data, or more specifically a block executes when data is made available at all of its inputs. Output data of the block are then sent to all other connected blocks. Data flow programming allows multiple operations to be performed in parallel since its execution is determined by the flow of data and not by sequential lines of code.

2.1 Virtual Instruments (VIs)

A VI consists of two major components; a front panel (FP) and a block diagram (BD). An FP provides the user-interface of a program, while a BD incorporates its graphical code. When a VI is located within the block diagram of another VI, it is called a subVI. LabVIEW VIs are modular, meaning that any VI or subVI can be run by itself.

2.1.1 Front Panel and Block Diagram

An FP contains the user interfaces of a VI shown in a BD. Inputs to a VI are represented by so-called controls. Knobs, pushbuttons and dials are a few examples of controls. Outputs from a VI are represented by so-called indicators. Graphs, LEDs (light indicators) and meters are a few examples of indicators. As a VI runs, its FP provides a display or user interface of controls (inputs) and indicators (outputs).

A BD contains terminal icons, nodes, wires, and structures. Terminal icons are interfaces through which data are exchanged between an FP and a BD. Terminal icons

correspond to controls or indicators that appear on an FP. Whenever a control or indicator is placed on an FP, a terminal icon gets added to the corresponding BD. A node represents an object which has input and/or output connectors and performs a certain function. SubVIs and functions are examples of nodes. Wires establish the flow of data in a BD. Structures such as repetitions or conditional executions are used to control the flow of a program. Figure 2-1 shows what an FP and a BD window look like.

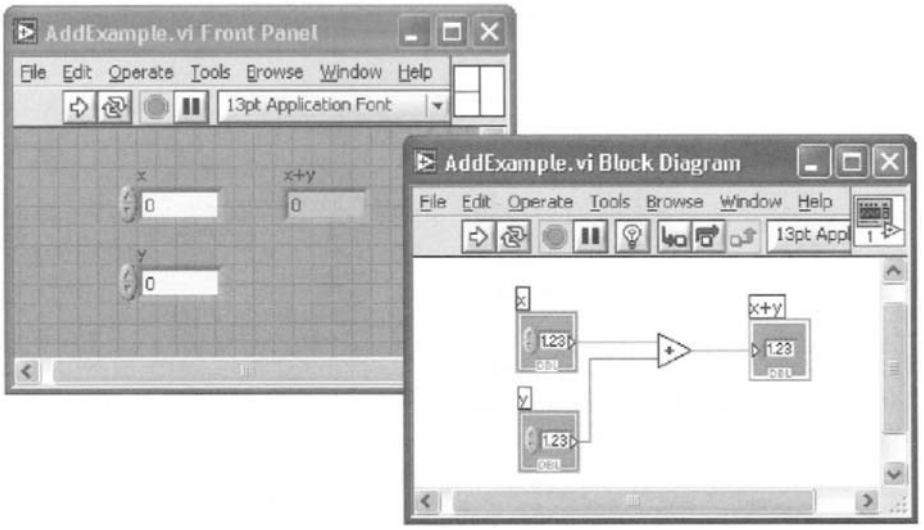


Figure 2-1: LabVIEW windows: front panel and block diagram.

2.1.2 Icon and Connector Pane

A VI icon is a graphical representation of a VI. It appears in the top right corner of a BD or an FP window. When a VI is inserted in a BD as a subVI, its icon gets displayed.

A connector pane defines inputs (controls) and outputs (indicators) of a VI. The number of inputs and outputs can be changed by using different connector pane patterns. In Figure 2-1, a VI icon is shown at the top right corner of the BD and its corresponding connector pane having two inputs and one output is shown at the top right corner of the FP.

2.2 Graphical Environment

2.2.1 Functions Palette

The Functions palette, see Figure 2-2, provides various function VIs or blocks for building a system. This palette can be displayed by right-clicking on an open area of a BD. Note that this palette can only be displayed in a BD.

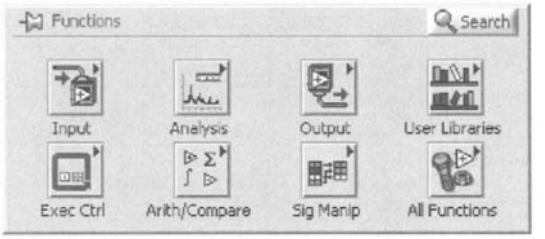


Figure 2-2: Functions palette.

2.2.2 Controls Palette

The Controls palette, see Figure 2-3, provides controls and indicators of an FP. This palette can be displayed by right-clicking on an open area of an FP. Note that this palette can only be displayed in an FP.

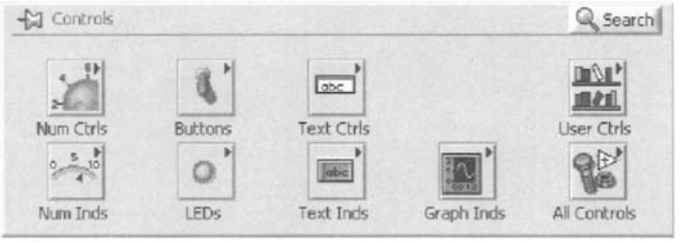


Figure 2-3: Controls palette.

2.2.3 Tools Palette

The Tools palette provides various operation modes of the mouse cursor for building or debugging a VI. The Tools palette and the frequently-used tools are shown in Figure 2-4.

Each tool is utilized for a specific task. For example, the Wiring tool is used to wire objects in a BD. If the automatic tool selection mode is enabled by clicking the **Automatic Tool Selection** button, LabVIEW selects the best matching tool based on a current cursor position.

Icon	Tool
	Automatic Tool Selection
	Operating tool
	Positioning tool
	Labeling tool
	Wiring tool
	Probe tool

Figure 2-4: Tools palette.

2.3 Building a Front Panel

In general, a VI is put together by going back and forth between an FP and a BD, placing inputs and outputs on the FP and building blocks on the BD.

2.3.1 Controls

Controls make up the inputs to a VI. Controls grouped in the Numeric Controls palette are used for numerical inputs, grouped in the Buttons & Switches palette for Boolean inputs, and grouped in the Text Controls palette for text and enumeration inputs. These control options are displayed in Figure 2-5.

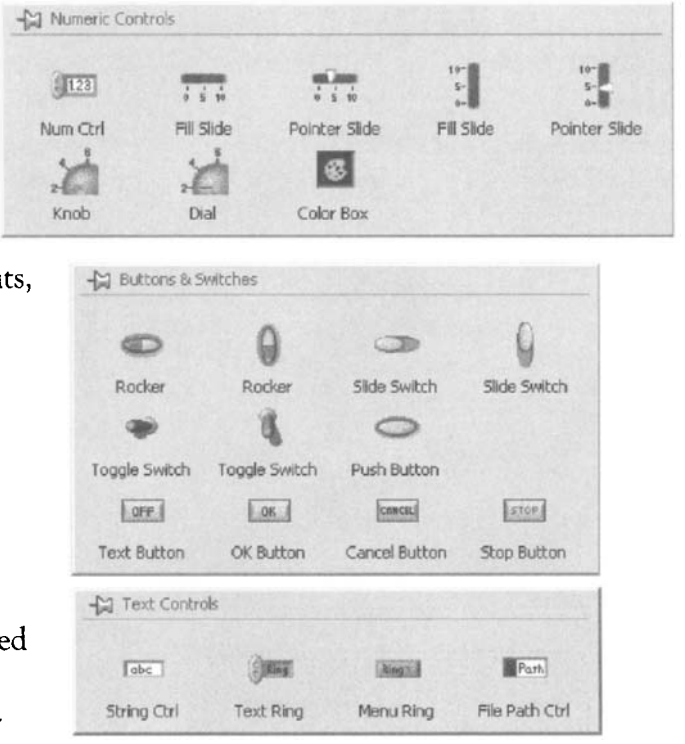


Figure 2-5: Control palettes.

2.3.2 Indicators

Indicators make up the outputs of a VI. Indicators grouped in the Numeric Indicators palette are used for numerical outputs, grouped in the LEDs palette for Boolean outputs, grouped in the Text Indicators

palette for text outputs, and grouped in the Graph Indicators palette for graphical outputs. These indicator options are displayed in Figure 2-6.

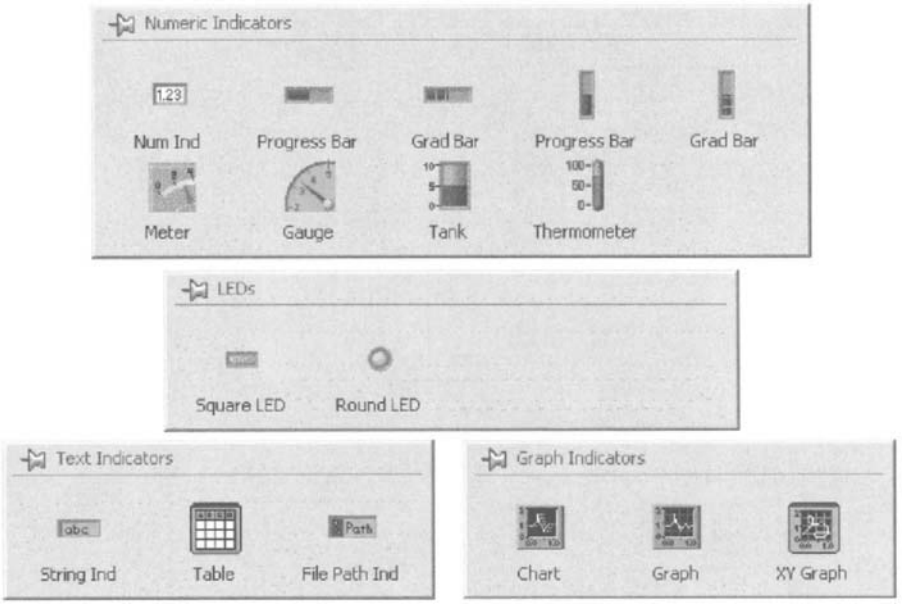


Figure 2-6: Indicator palettes.

2.3.3 Align, Distribute and Resize Objects

The menu items on the toolbar of an FP, see Figure 2-7, provide options to align and distribute objects on the FP in an orderly manner. Normally, after controls and indicators are placed on an FP, one uses these options to tidy up their appearance.

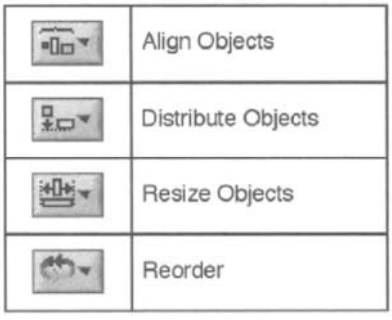


Figure 2-7: Menu for align, distribute, resize, and reorder objects.

2.4 Building a Block Diagram

2.4.1 Express VI and Function

Express VIs denote higher-level VIs that have been configured to incorporate lower-level VIs or functions. These VIs are displayed as expandable nodes with a blue background. Placing an Express VI in a BD brings up a configuration dialog window allowing adjustment of its parameters. As a result, Express VIs demand less wiring. A configuration window can be brought up by double-clicking on its Express VI.

Basic operations such as addition or subtraction are represented by functions. Figure 2-8 shows three examples corresponding to three types of a BD object (VI, Express VI, and function).

Figure 2-8: Block Diagram objects: (a) VI, (b) Express VI, and (c) function.

Both subVI and Express VI can be displayed as icons or expandable nodes. If a subVI is displayed as an expandable node, the background appears yellow. Icons are used to save space in a BD, while expandable nodes are used to provide easier wiring or better readability. Expandable nodes can be resized to show their connection nodes more clearly. Three appearances of a VI/Express VI are shown in Figure 2-9.

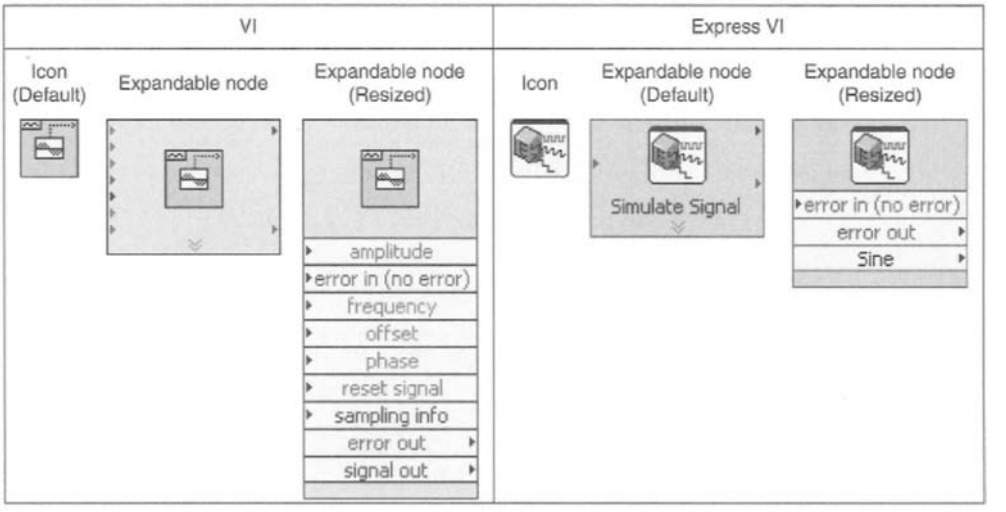


Figure 2-9: Icon versus expandable node.

2.4.2 Terminal Icons

FP objects are displayed as terminal icons in a BD. A terminal icon exhibits an input or output as well as its data type. Figure 2-10 shows two terminal icon examples consisting of a double precision numerical control and indicator. As shown in this figure, terminal icons can be displayed as data type terminal icons to conserve space in a BD.

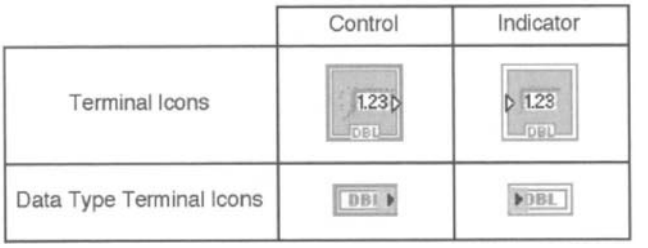


Figure 2-10: Terminal icon examples displayed in a BD.

2.4.3 Wires

Wires transfer data from one node to another in a BD. Based on the data type of a data source, the color and thickness of its connecting wires change.

Wires for the basic data types used in LabVIEW are shown in Figure 2-11. Other than the data types shown in this figure, there are some other specific data types. For example, the dynamic data type is always used for Express VIs, and the waveform data type, which corresponds to the output from a waveform generation VI, is a special cluster of waveform components incorporating trigger time, time interval, and data value.

Wire Type	Scalar	1D Array	2D Array	Color
Numeric				Orange (Floating point) Blue (Integer)
Boolean				Green
String				Pink

Figure 2-11: Basic types of wires [2].

2.4.4 Structures

A structure is represented by a graphical enclosure. The graphical code enclosed by a structure is repeated or executed conditionally. A loop structure is equivalent to a for loop or a while loop statement encountered in text-based programming languages, while a case structure is equivalent to an if-else statement.

2.4.4.1 For Loop

A For Loop structure is used to perform repetitions. As illustrated in Figure 2-12, the displayed border indicates a For Loop structure, where the count terminal N represents the number of times the loop is to be repeated. It is set by wiring a value from outside of the loop to it. The iteration terminal i denotes the number of completed iterations, which always starts at zero.

Figure 2-12: For Loop.

2.4.4.2 While Loop

A While Loop structure allows repetitions depending on a condition, see Figure 2-13. The conditional terminal initiates a stop if the condition is true. Similar to a For Loop, the iteration terminal i provides the number of completed iterations, always starting at zero.

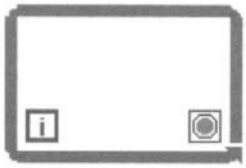


Figure 2-13: While Loop.

2.4.4.3 Case Structure

A Case structure, see Figure 2-14, allows running different sets of operations depending on the value it receives through its selector terminal, which is indicated by ?. In addition to Boolean type, the input to a selector terminal can be of integer, string, or enumerated type. This input determines which case to execute. The case selector True shows the status being executed. Cases can be added or deleted as needed.

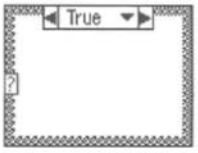


Figure 2-14: Case structure.

2.5 Grouping Data: Array and Cluster

An array represents a group of elements having the same data type. An array consists of data elements having a dimension up to $2^{31}-1$. For example, if a random number is generated in a loop, it makes sense to build the output as an array since the length of the data element is fixed at 1 and the data type is not changed during iterations.

A cluster consists of a collection of different data type elements, similar to the structure data type in text-based programming languages. Clusters allow one to reduce the number of wires on a BD by bundling different data type elements together and passing them to only one terminal. An individual element can be added to or extracted from a cluster by using the cluster functions such as `Bundle by Name` and `Unbundle by Name`.

2.6 Debugging and Profiling VIs

2.6.1 Probe Tool

The Probe tool is used for debugging VIs as they run. The value on a wire can be checked while a VI is running. Note that the Probe tool can only be accessed in a BD window.

The Probe tool can be used together with breakpoints and execution highlighting to identify the source of an incorrect or an unexpected outcome. A breakpoint is used to pause the execution of a VI at a specific location, while execution highlighting helps one to visualize the flow of data during program execution.

2.6.2 Profile Tool

The Profile tool can be used to gather timing and memory usage information, in other words, how long a VI takes to run and how much memory it consumes. It is necessary to make sure a VI is stopped before setting up a Profile window.

An effective way to become familiar with LabVIEW programming is by going through examples. Thus, in the two labs that follow in this chapter, most of the key programming features of LabVIEW are presented by building some simple VIs. More detailed information on LabVIEW programming can be found in [1-5].

2.7 Bibliography

[1] National Instruments, *LabVIEW Getting Started with LabVIEW*, Part Number 323427A-01, 2003.

[2] National Instruments, *LabVIEW User Manual*, Part Number 320999E-01, 2003.

[3] National Instruments, *LabVIEW Performance and Memory Management*, Part Number 342078A-01, 2003.

[4] National Instruments, *Introduction to LabVIEW Six-Hour Course*, Part Number 323669B-01, 2003.

[5] Robert H. Bishop, *Learning With Labview 7 Express*, Prentice Hall, 2003.

Lab 1: Getting Familiar with LabVIEW: Part I

The objective of this first lab is to provide an initial hands-on experience in building a VI. For detailed explanations of the LabVIEW features mentioned here, the reader is referred to [1]. LabVIEW7.1 can get launched by double-clicking on the LabVIEW 7.1 icon. The dialog window shown in Figure 2-15 should appear.

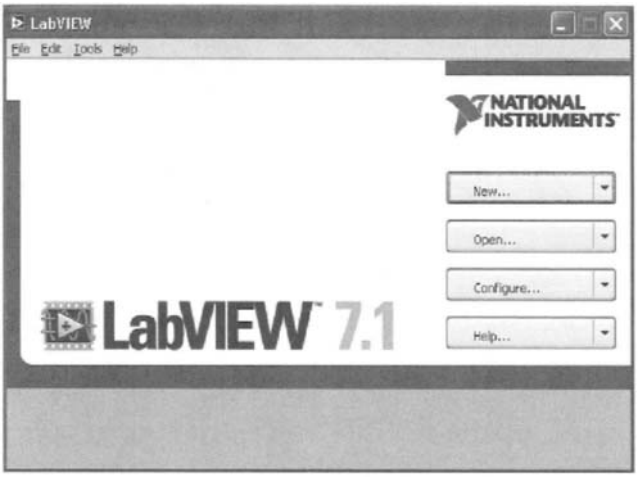


Figure 2-15: Starting LabVIEW.

L1.1 Building a Simple VI

To become familiar with the LabVIEW programming environment, it is more effective if one goes through a simple example. The example presented here consists of calculating the sum and average of two input values. This example is described in a step-by-step fashion below.

L1.1.1 VI Creation

To create a new VI, click on the arrow next to **New**... and choose **Blank VI** from the pull down menu. This step can also be done by choosing **File** → **New VI** from the menu. As a result, a blank FP and a blank BD window appear, as shown in Figure 2-16. It should be remembered that an FP and a BD coexist when building a VI.

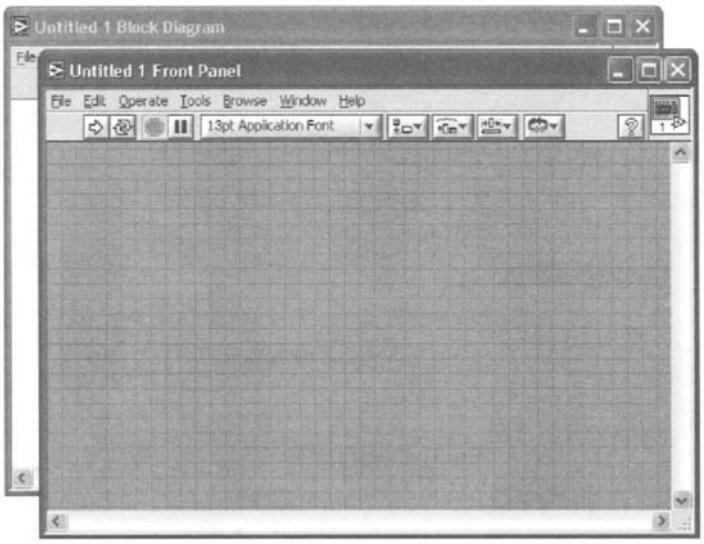


Figure 2-16: Blank VI.

Clearly, the number of inputs and outputs to a VI is dependent on its function. In this example, two inputs and two outputs are needed, one output generating the sum and the other the average of two input values. The inputs are created by locating two `Numeric Controls` on the FP. This is done by right-clicking on an open area of the FP to bring up the Controls palette, followed by choosing **Controls → Numeric Controls → Numeric Control**. Each numeric control automatically places a corresponding terminal icon on the BD. Double-clicking on a numeric control highlights its counterpart on the BD, and vice versa.

Next, let us label the two inputs as `x` and `y`. This is achieved by using the Labeling tool from the **Tools** palette, which can be displayed by choosing **Window → Show Tools Palette** from the menu bar. Choose the Labeling tool and click on the default labels, `Numeric` and `Numeric 2`, in order to edit them. Alternatively, if the automatic tool selection mode is enabled by clicking **Automatic Tool Selection** in the **Tools** palette, the labels can be edited by simply double-clicking on the default labels. Editing a label on the FP changes its corresponding terminal icon label on the BD, and vice versa.

Similarly, the outputs are created by locating two `Numeric Indicators` (**Controls → Numeric Indicators → Numeric Indicator**) on the FP. Each numeric indicator automatically places a corresponding terminal icon on the BD. Edit the labels of the indicators to read `Sum` and `Average`.

For a better visual appearance, objects on an FP window can be aligned, distributed, and resized using the appropriate buttons appearing on the FP toolbar. To do this,

select the objects to be aligned or distributed and apply the appropriate option from the toolbar menu. Figure 2-17 shows the configuration of the FP just created.

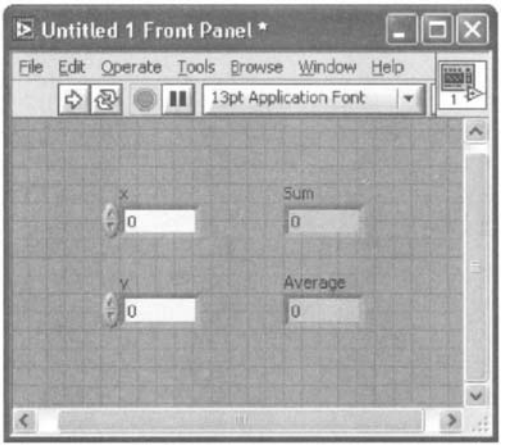
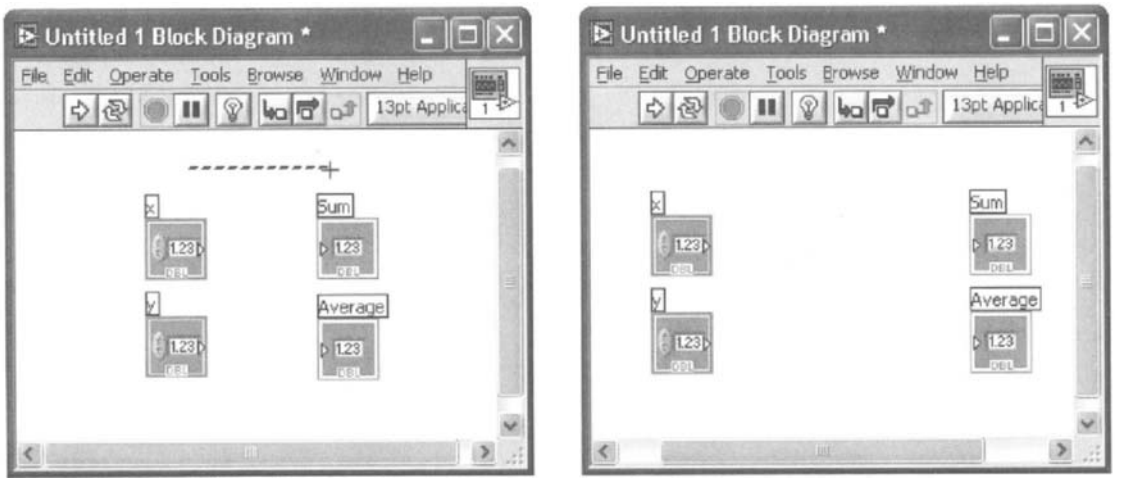


Figure 2-17: FP configuration.

Now, let us build a graphical program on the BD to perform the summation and averaging operations. Note that <Ctrl + E> toggles between an FP and a BD window. If one finds the objects on a BD are too close to insert other functions or VIs in between, a horizontal or vertical space can be inserted by holding down the <Ctrl> key to create space horizontally and/or vertically. As an example, Figure 2-18 (b) illustrates a horizontal space inserted between the objects shown in Figure 2-18 (a).



(a) (b)

Figure 2-18: Inserting horizontal/vertical space: (a) creating space while holding down the <Ctrl> key, and (b) inserted horizontal space.

Next, place an `Add` function (**Functions → Arithmetic & Comparison → Express Numeric → Add**) and a `Divide` function (**Functions → Arithmetic & Comparison → Express Numeric → Divide**) on the BD. The divisor, in our case 2, needs to be entered in a `Numeric Constant` (**Functions → Arithmetic & Comparison → Express Numeric → Numeric Constant**) and connected to the y terminal of the `Divide` function using the Wiring tool.

To have a proper data flow, functions, structures and terminal icons on a BD need to be wired. The Wiring tool is used for this purpose. To wire these objects, point the Wiring tool at a terminal of a function or a subVI to be wired, left click on the terminal, drag the mouse to a destination terminal and left click once again. Figure 2-19 illustrates the wires placed between the terminals of the numeric controls and the input terminals of the add function. Notice that the label of a terminal is displayed whenever the cursor is moved over it if the automatic tool selection mode is enabled. Also, note that the Run button on the toolbar remains broken until the wiring process is completed.

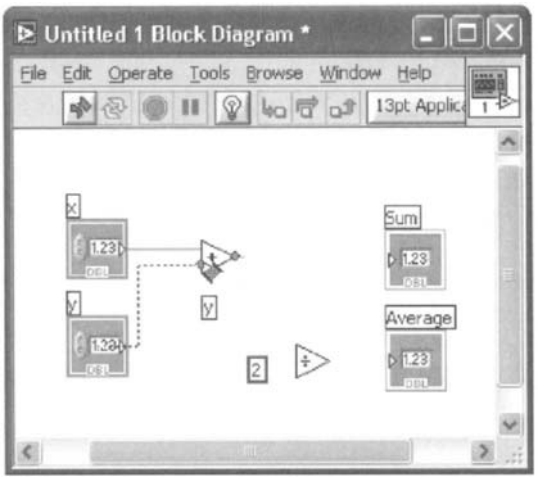


Figure 2-19: Wiring BD objects.

For better readability of a BD, wires which are hidden behind objects or crossed over other wires can be cleaned up by right-clicking on them and choosing **Clean Up Wire** from the shortcut menu. Any broken wires can be cleared by pressing <Ctrl + B> or **Edit → Remove Broken Wires**.

The label of a BD object, such as a function, can be shown (or hidden) by right-clicking on the object and checking (or unchecking) **Visible Items → Label** from the shortcut menu. Also, a terminal icon corresponding to a numeric control or indicator